

45
185 pfs.
microproces

ALUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek



LOGO



¡ LOGO es un lenguaje, que en sus algo más de diez años de existencia está teniendo importancia para la introducción en el mundo informático

de los no iniciados. Precisamente Seymour Papert pensó en éstos al crear este lenguaje, concretamente en los niños. Una de las principales características que presenta es que aquel que no haya visto nunca un ordenador puede realizar programas en LOGO en muy poco tiempo, algo impensable en un lenguaje tan popular como el BASIC.

Por lo que llevemos dicho hasta ahora podríamos estar tentados a pensar que entonces este lenguaje es de segunda categoría y que sólo es «juego de niños». Al contrario, el LOGO es un lenguaje que presenta algunas características novedosas sobresalientes (como son los gráficos de tortuga que más adelante veremos) que han sido incorporados actualmente en otros lenguajes considerados como más potentes.

Pero ¿cuáles son esas características que hacen que el LOGO sea un lenguaje de primera categoría pero apropiado para el no iniciado en este mundo? Se pueden resumir en los siguientes:

— Es un lenguaje informático que suele ser presentado en el idioma en que se expresa el usuario, por lo tanto éste tendrá menos problemas en aprenderse los comandos (mejor llamados primitivos) y en saber sus funciones.

— Lleva incorporado uno de los sistemas más simples pero al mismo tiempo más potentes para la realización de dibujos, los muy conocidos «gráficos de tortuga». Con éstos se pueden realizar dibujos muy complicados utilizando muy pocos comandos.

— Maneja de forma muy sencilla palabras y frases, que pueden estar formadas por números, palabras u otras frases, lo que puede ser equivalente en BASIC a las cadenas de caracteres y cuya potencia en LOGO hace que se vean diferentes



Además de las manijas de listas y de los gráficos de tortuga, algunos recursos LOGO pueden realizar operaciones aritméticas y trigonométricas.

— Perteneció al grupo de los lenguajes llamados estructurados, es decir que utiliza procedimientos (bloques de programa que realiza una función concreta, equivalente en BASIC a las subrutinas llamadas por GOSUB) e incluso con posibilidad de recursividad, es decir, que un procedimiento se puede llamar a sí mismo.

— Si a todo esto se le añade que muchas versiones LOGO incorporan funciones normales en otros lenguajes como son las operaciones aritméticas y las razones trigonométricas, hace que nos encontremos con un lenguaje para sentirse tran-

!!

No hay que pensar equivocadamente si vamos a trabajar con símbolos matemáticos o álgebra infinitos.

*

Para ejecutar un procedimiento se escribe su nombre como a se tratase de una primitiva incorporada.

*

Para trabajar con los gráficos se supone que tenemos una tortuga (ese mariguato que aparece en la pantalla) dotado de un lápiz, le cual dibujó el trazo donde y cuando nosotros queramos.



LOGO es un lenguaje pensado para la educación.



Los procedimientos son bloques de programa independientes que cada uno de ellos realiza una función

quilarmente ante nuestro «micro» y descubrir sus intencionalidades.

Y precisamente esto es lo que vamos a hacer. Cogemos nuestro SPECTRUM y el LOGO CASTELLANO de LCSU/SOLU y bucear en las posibilidades que nos ofrece.

VARIABLES Y PROCEDIMIENTOS

En líneas generales el LOGO no trabaja de una manera diferente al BASIC con las variables, aunque sí cambia, como es de suponer, los comandos y la sintaxis. Podemos realizar el simulé de suponer que una variable es como una caja en la cual guardamos la información que nos interesa. Para encontrar pronto lo que tenemos guardado en la caja, a ésta la ponemos un nombre que nos identifique el contenido de su interior, pues igualmente a una variable hay que ponerle un nombre para identificar lo que queremos guardar.

¿Ponemos un ejemplo? Supongamos que en nuestra caja queremos guardar los meses del año apuntados en una hoja de papel, por supuesto que el mejor título que podemos poner es el de «meses», y, entonces, siguiendo con el simulé, a nuestra variable la llamaremos también «meses». Vamos a guardar el primer dato en nuestra «caja/variable». Para ello cogemos una hoja de papel y escribimos en ella lo que nos interesa, por ejemplo «enero», y lo colocamos dentro de la caja con la precaución de dejar el dato escrito boca arriba. Pero guardar ese dato en nuestra variable

«meses» en nuestro LOGO lo haremos de la siguiente manera:

? ASIGNA "meses" "enero"

Ahora ya tenemos la seguridad de que nuestro dato no se nos perderá y sabemos que siempre que acudamos a "meses" nos encontraremos lo mismo que hemos guardado. En el caso de la caja nos basta con asomarnos y echar una mirada en su interior para saber qué es lo que tenemos guardado, pero, claro está, con el LOGO eso no lo podemos hacer. Le tendremos que decir a nuestro «micro» que por pantalla nos muestre el valor de la variable "meses".

? IMPRIME "meses"

Y en pantalla aparecerá:

enero

Sin embargo, el LOGO nos ofrece también la posibilidad de asignar a una variable no una sino una serie de palabras (lo que se conoce como listas). Sería el equivalente a que en nuestra caja introduzásemos una hoja en la que se han escrito varios meses. Veamos cómo se hace en LOGO.

?ASIGNA "meses (enero febrero marzo abril)

Si entonces hacemos:

?IMPRIME meses

En pantalla aparecerá, por tanto:

enero febrero marzo

A las variables se las puede considerar como cajas con su contenido especificado en el exterior y en el cual introduzcamos la palabra a guardar.



!!

El LOGO nos ofrece también la posibilidad de asignar a una variable palabras y listas.

*

Llévate incorporado uno de los sistemas más amplios para el mismo tiempo más potentes para la realización de dibujos gráficos de tortuga.

Aunque aquí hemos trabajado con variables alfanuméricas, con números se hace exactamente igual, al LOGO no hay que avisarle previamente si vamos a trabajar con uno u otro tipo de variable. Por tanto, el lector comprenderá perfectamente los siguientes órdenes y sabrá que aparecerá en la pantalla:

```
?ASIGNA "das 30
?ASIGNA "das (20 29 30 31)
```



Aunque el LOGO es un lenguaje para las que se hacen en informática no debe ser considerado de segunda categoría

NUESTRO PRIMER PROGRAMA

Sin embargo, programar así no aporta ninguna ventaja a no ser que podamos agrupar una serie de comandos para resolver un problema concreto. Es aquí donde entra en juego los procedimientos.

```
?PARA media
>IMPRIME (8+6)/2
?FIN
```

Aquí tenemos nuestro primer programa en LOGO que consta de un sólo procedimiento, al cual le hemos llamado «media». Para que nuestro intérprete LOGO distinga si estamos introduciendo un comando directo o definiendo un procedimiento se emplea la primitiva «PARA». Para indicar el final de dicho procedimiento se escribe «FIN», a lo cual el ordenador nos contestará, para indicar que se ha dado por terminado, con el siguiente mensaje:

media DEFINIDO

Para ejecutar nuestro procedimiento se escribe su nombre como si se tratase de una primitiva más incorporada:

```
?media
?
```

Sin embargo, este procedimiento nos sirve de muy poco puesto que sólo nos calcula la media de 8 y 6, para que sea verdaderamente útil y se pueda calcular la media aritmética de dos números cualesquiera deberíamos tener el siguiente procedimiento:

```
?PARA media2a :primum :segnum
>IMPRIME (:primum + :segnum)/2
?FIN
```

Para ejecutar este procedimiento introduciremos el nombre y los valores que queramos tomar las variables en cada caso:

```
?media2a 8 2
?
```

Es evidente que el valor 8 se asigna a «:primum» y el de 2 a «:segnum».

GRAFICOS DE TORTUGA

Como ya dijimos al principio del presente capítulo, una de las características más sobresalientes del LOGO es la posibilidad de realizar dibujos de una forma bastante sencilla. Para ello se supone que disponemos de una tortuga (ese triángulo que aparece en medio de la pantalla señalando hacia arriba cuando comenzamos a trabajar con

En una variable también podemos guardar no sólo palabras sino una lista de ellas.



!

Podemos realizar una asignación a una variable de una lista formada a su vez por otro tipo además de palabras.

*

Las listas podemos modificarlas (añadiendo palabras o borrar), imprimirles totalmente o imprimir cualquier parte de ella.

*

Maneja de forma muy sencilla palabras y listas, que a su vez pueden estar formadas por números, palabras o otros tipos.

CON LAPIZ
GIRA DERECHA 90
AVANZA 70



A nuestra tortuga le podemos decir que levante y baje el
lápiz cuando nos convenga.

LOGO) muy bien amatestrada y dotada de un lá-
piz el cual dejará el trazo dando y cuando noso-
tros queramos. Así el siguiente programa la ha-
cemos subir y bajar el lápiz a nuestra amiga,
?PARA paralelas
>AVANZA 70
>SINLAPIZ GIRADERECHA 90 AVANZA 30
>CONLAPIZ GIRADERECHA 90 AVANZA 70
>FIN

Para ejecutar un procedimiento sin variables basta con
dar el nombre de dicho procedimiento.

En el manual del intérprete de LOGO viene un
ejemplo de como hacer un cuadrado. Para evitar
repetirnos vamos a ir mas allá y aquí presenta-
mos un procedimiento para realizar cualquier fi-
gura regular.

?PARA regular: lado
>BORRAPANTALLA
>ASIGNA "angulo 360/:lado
>REPITE :lado (AVANZA 10 GIRADERECHA
.angulo)
>FIN

Como hemos podido comprobar, la comprensión
de los procedimientos y de la función que ellos
tienen es bien fácil ya que se utilizan primitivas
escogidas de vocabulario usual, y ante las que
nosra amiga la tortuga se doblaba y cum-
ple la función encomendada con su lápiz a oren-
tas dejando la huella de su recorrido.



Uno de los procedimientos que presenta el LOGO a prar
de su potencia es su flexibilidad.

Para finalizar esta sección dedicada a los gráfi-
cos vamos a dejar un procedimiento que realice
un dibujo complicado en comparación con lo que
ocupa dicho procedimiento. Esto simplemente es
una muestra de la originalidad a que puede dar
lugar el LOGO con muy poco esfuerzo. ¿A qui no
nos atrevamos a hacer el siguiente programa
en BASIC?

?PARA giras
>REPITE 4 (AVANZA 50 GIRADERECHA 90)
>GIRADERECHA 35
>giras
>FIN

Este procedimiento, como se puede observar, se
llama a sí mismo incondicionalmente, con lo cual
estará permanentemente ejecutándose a no ser
que se impida con BREAK.



LAS PALABRAS Y LAS LISTAS

Otro de los aspectos importantes del LOGO (y del cual no se mencionó nada en el manual LOGO de LC3V/SOL) es el manejo de palabras y listas.

ASIGNA "países (japón españa francia)

En este caso en la variable «países» hemos introducido una lista formada todo ella de palabras. También podemos realizar una asignación a una variable de una lista formada a su vez por otra u otras listas además de palabras.

ASIGNA "paíseseibis (europa (españa francia italia) japon)

Ahora en la variable «paíseseibis» hay una lista formada por otra lista, la que se llama europa, que está a su vez formada por palabras.

Las listas podemos modificarlas, imprimir las totalmente o imprimir sólo la parte de ellas. Por ejemplo:

IMPRIME PRIMERO países

japón

IMPRIME ULTIMO países

francia

En el primer caso hemos tomado el primero de la lista dentro de la variable «países» y en el segundo nos aparecerá en pantalla el último de la lista. Pero, ¿qué pasa con los datos centrales?, ¿no podemos acceder a ellos? Ahora veremos cómo.

IMPRIME MENOSPRIMERO países

españa francia

IMPRIME MENOSULTIMO países

japón españa

El LOGO está provisto de un lápiz con el cual va dejando su huella para hacer los dibujos.

Ya hemos accedido a la parte central de la lista, pero todavía seguimos teniendo pegado el primer y último elemento, de la lista, pero ya tenemos bastantes herramientas para coger cualquier palabra de nuestra lista.

IMPRIME PRIMERO MENOSPRIMERO

países

españa

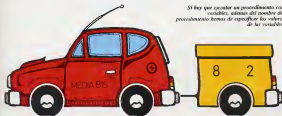
IMPRIME ULTIMO MENOSULTIMO países

españa

Lo que nos hace el primer caso es imprimir la primera palabra de la lista una vez quitado el primero, resumiendo, nos imprime el segundo. Igualmente, el siguiente caso nos imprime el último elemento una vez quitado el último, es decir, nos imprime el penúltimo igualmente, para seguir teniendo acceso a los datos más interiores de la lista basta con añadir tantos "MENOSPRIMERO" y "MENOSULTIMO" como sea necesario.

Podemos añadir también datos a una lista, no sólo extraerlos. Para ello utilizaremos los comandos **POINPR** (abreviatura de **POINer PRIMero**) y **PONULT** (abreviatura de **POINer ULTIMo**).

Si hay que prestar un procedimiento con variables, además del nombre del procedimiento hemos de especificar los valores de las variables.



!

Uno de los incrementos del lenguaje LOGO es su longitud para hacer un programa largo y complejo.

*

El LOGO es un lenguaje que presenta algunas características novedosas sobresalientes que han sido incorporadas actualmente en otros lenguajes considerados como más potentes.

Una lista es una serie de palabras en la cual hay que distinguir el primero y el último



IMPRIIME PONRI "mexico :países
mexico
IMPRIIME PONULT "portugal :países
portugal

PARA FINALIZAR

Con el espacio limitado que ofrece este artículo no hay lugar para profundizar más en este lenguaje. Ahora es la originalidad del lector la que se pone a prueba para que ponga en práctica toda su habilidad para realizar programas. Se puede ayudar, para ello, el manual LOGO, que salvo algunas carencias (no menciona cómo trabajar con listas y palabras y no explica cómo utilizar el ZX MICRODRIVE, ya que se puede trabajar con él, es fácil de entender. Además, al final del mismo

tiene un glosario de todas las palabras primitivas que utiliza y unas indicaciones de su sintaxis y de su función.

Creo que ha quedado claro que este lenguaje no puede ser considerado de segunda categoría aunque sea el más indicado para aprender. Aporta ventajas importantes, como el uso de los procedimientos, algo que el BASIC no presenta y que lo relaciona con los GOTOs y GOSUBs pero que en un programa largo complica la comprensión y el seguimiento del mismo. Será difícil que un iniciado atrape «malos visos» con el LOGO y muy fácil que lo haga con el BASIC, que luego además serán traspasables a otros lenguajes con los problemas que ocasiona, sobre todo porque la mayor parte de los lenguajes informáticos utilizan procedimientos.

Uno de los inconvenientes del lenguaje LOGO es su lentitud para hacer un programa largo y complicado (se puede ganar algo de tiempo en los gráficos de tortuga escondiendo éste). Sin embargo, el objetivo de este lenguaje no es su velocidad sino el abrir las puertas a aquellos que quieren adentrarse en el campo informático. Si es necesario realizar programas más complicados hay que utilizar otros lenguajes más potentes, como podría ser el PASCAL.



PONER PRIMERO

PONER ULTIMO



En una lista podemos añadir elementos por delante y detrás.

DE 8 EN 8 BITS



entadas ya las bases previas necesarias para la comprensión de los distintos grupos de instrucciones, comenzamos en este capítulo a realizar un análisis detallado de cada uno de ellos. A partir de la misión concreta de una determinada instrucción (discutiremos siempre en el texto su funcionamiento), encontraremos otros datos de interés para la programación en C/M en las tablas que presentamos como desglose de cada grupo particular, cuyo significado y utilidad discutimos en el bloque anterior, tales como los ciclos de reloj necesarios para su ejecución o el estado del registro de indicadores F.

REGISTRO	DECODIFICACION r o r'
A	111
B	000
C	001
D	010
E	011
H	100
L	101

LD DESTINO, ORIGEN

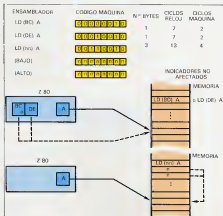
Volvemos a las tablas del capítulo 41. En ellas, bajo la denominación «Carga de ocho bits» encontramos una colección de instrucciones, todas ellas de código mnemónico LD (abreviatura del inglés Load, cargar) encargadas de transferir los movimientos de información que involucran a registros, posiciones de memoria y datos individuales.

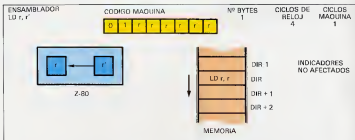
Todas ellas responden al formato

LD Destino, Origen o LD Destino, Dato

donde tanto Origen como Destino pueden ser registros o posiciones de memoria individuales y Dato un valor numérico almacenado en un octeto, y por tanto, en el rango de 0 a 255. Para determinar el código de operación de una de estas instrucciones entraremos en la tabla por la fila marcada con DESTINO y buscaremos la información con la columna ORIGEN, lugar en el cual encontraremos los códigos hexadecimales correspondientes.

Si allí aparece un cuadro en blanco, la instrucción planteada no está contemplada dentro del «juego» presente en el microprocesador Z 80. El resultado de una instrucción de «Carga» trae





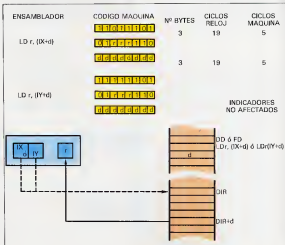
consegua la «copia» del contenido de la posición de memoria o registro Origen en el registro o posición de memoria Destino, manteniéndose SIEMPRE inalterado el primero por la ejecución de la instrucción (en su momento comentaremos una excepción, la cual confirma la regla).

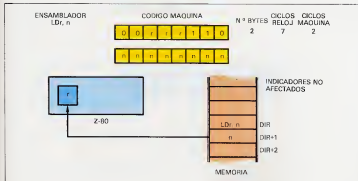
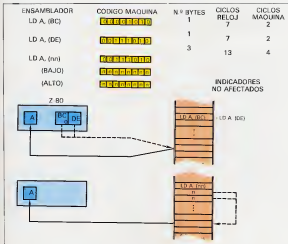
Efectuando un recorrido por la tabla correspondiente es factible la siguiente clasificación:

1) Carga en registros

1.1 Entre registros: LD r, r'

Con r y r' representamos cualquiera de los regis-





tros A, B, C, D, E, H o L y en la figura encontramos la decodificación seguida por el Z 80 para reconocerlos, siendo el formato general de esta instrucción en código máquina:

01(-r)(-r')

Consideremos, por ejemplo, la instrucción LD A,B perteneciente a este grupo. En las tablas encontramos su código de operación asociado: 78h. Utilizando la decodificación dada en la figura obtenemos el byte 01111000, el cual no es otra cosa que la representación binaria del código anterior. Para su ejecución, el microprocesador necesita realizar una sola lectura de la memoria, pues toda la información que precisa está en el propio código de operación. Por tanto, empleará 4 ciclos de reloj, no siendo afectado por esta instrucción el registro de indicadores.

1.2. Carga inmediata: LD r,n

Mediante esta instrucción se almacena en el registro r, el valor numérico especificado en los 8 bits del dato n. Para ello son necesarios dos ciclos máquina: uno para localizar el código de operación

(4 ciclos de reloj) y otro para leer el dato (3 ciclos) sin cambio en los indicadores de estado. Los dos bytes que la componen ofrecen el siguiente aspecto:

00(-r)-1110
(-n...)

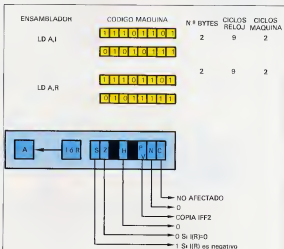
1.3. Carga indirecta: LD r,(HL)

Estas instrucciones recurren al par de registros HL como puntero de una determinada posición de memoria, donde leer el byte a ser copiado en el registro r.

La ejecución del único octeto que la conforma


01(-r)-1110

no influye en los indicadores de condición, siendo necesarios 2 ciclos de memoria, uno para la búsqueda del código de operación y otro para la lectura del dato señalado por HL. En total 7 ciclos de reloj como ya sabemos calcular. Idéntico comportamiento y comentarios son factibles sobre las instrucciones LD A,(BC) y LD A,(DE). Como observaremos en ellas r debe ser



ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD I, A	1 1 1 0 1 1 0 1	2	9	2
	0 1 0 0 0 1 1 1			
LD R, A	1 1 1 0 1 1 0 1	2	9	2
	0 1 0 0 1 1 1 1			

Z-80



INDICADORES
NO AFECTADOS

inevitablemente el acumulador A, pues de otra manera, no están contemplados en el juego del Z-80.

En este mismo grupo podríamos incluir una instrucción, la cual exclusivamente trabaja sobre el acumulador sirviéndose de direccionamiento inmediato: LD A, (nn).

Mediante ella se almacena en éste el contenido de la posición de memoria definida por los dos octetos nn, debiéndose especificar en primer lugar el byte bajo de tal dirección (recordemos que cuando aparecen los paréntesis en una instrucción escrita en ensamblador debe leerse el contenido de « »).

Empieza 13 ciclos de reloj repartidos, 4 para buscar el código de operación, 3 más 3, para leer los dos octetos nn, y finalmente otros 3 al buscar el dato contenido en la posición nn. Los indicadores no se ven afectados.


1.4. Carga Indexada: LD r, (IX+d) y LD r, (IY+d)
El objetivo de estas instrucciones es colocar en el registro r el contenido de la posición de memoria obtenida al sumar el valor actual del par de registros índice (IX o IY (según empleemos la primera o la segunda) y un byte de desplazamiento d expresado en complemento a dos, y por tanto, en el rango -128 a +127.

Aclaremoslo con un ejemplo: supongamos que deseamos ejecutar la instrucción LD B, (IX-10) y que en ese momento el par IX contiene 32767 (7FFFh). Busquemos en la tabla los códigos hexadecimales asociados donde encontraremos 00 48 d. Nos queda, por tanto, sustituir d por su valor en dicho sistema de numeración.


Pues su valor es F8. ¿Por qué? Recordemos que el desplazamiento hemos de expresarlo en complemento a dos, y precisamente F8 es la repre-

ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD (HL), r	0 0 0 0 0 0 0 0	1	7	2
LD r, (HL)	0 0 0 0 0 0 0 0	1	7	2
LD (HL), nn	0 0 0 0 0 0 0 0	2	10	3


Z-80



Z-80



Z-80



INDICADORES NO AFECTADOS

MEMORIA

MEMORIA

MEMORIA

sentación en este sistema del número decimal -10. Con ésta la instrucción completa es **DD 46 F6**. ¿Qué ocurre al ejecutarla?

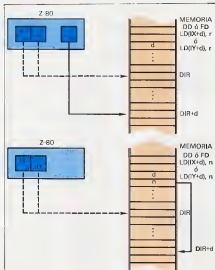
El microprocesador buscará el valor del par de registros **IX** (32767 en el ejemplo). A continuación, restará 10 unidades de éste obteniendo 32757. Finalmente, extraerá los ocho bits almacenados en esta posición de memoria y los cargará en el registro **B**.

Podemos comprobar en las tablas que el primer byte de todas las instrucciones que utilizan el registro **IX** es **DD**, y el de las que manejan al **IX**, **FD**.

2) Carga en memoria

En cierto modo, las instrucciones del presente grupo pueden considerarse como inversas de las anteriores, al menos si tenemos en cuenta que el papel desempeñado por los registros o posiciones de memoria **Origen** y **Destino** afectados se intercambia respecto al sentido en que se efectúa la transferencia de datos.

2.1 Carga indirecta: **LD (HL), r** y **LD (HL), a**



Al ejecutar estas instrucciones se consigue almacenar en el octeto de memoria especificado por el par de registros **HL**, el valor actual del registro **r** o el dato **n**. Veámos a registrar en que tras el proceso, tanto **HL**, como **r** o **n** se mantienen inalterados.

Dentro de este mismo apartado es posible considerar tres instrucciones las cuales sólo tienen sentido cuando el registro **Origen** es el acumulador **A**.

LD (BC), A **LD (DE), A** **LD (nn), A**

El funcionamiento es idéntico a los anteriores, pero el contenido de **A** es transfiriendo a la dirección de memoria señalada por **BC**, **DE** o los dos bytes **nn**. Recuerda que al utilizar la última, los octetos no deberán especificarse en el orden bajo-alto.

2.2 Carga indirecta indexada: **LD (IX+d), r** **LD (IX+d), n** **LD (IY+d), r** **LD (IY+d), n**

Para establecer la posición de memoria donde almacenar el contenido del registro **r** o el dato **n**, el microprocesador sigue el mismo sistema que explicásemos cuando nos referíamos en el grupo 1.4 a las instrucciones que manejan direccionamiento indexado.

3) Carga sobre registros especiales: **LD A, I** **LD A, R** **LD I, A** **LD R, A**

Manejando los dos primeros conseguimos trasladar al acumulador el contenido del vector de página de interrupción **I**, o el valor del registro de regeneración de memoria **R**, respectivamente. Son las únicas instrucciones dentro del grupo de carga de 8 bits que alteran el contenido del registro de indicadores. Veámos cuál es el efecto sobre ellos.

C No es afectado por estas instrucciones.

Z Se coloca a 0 si **I (R)** vale 0.

S Encontraremos un 1 cuando el bit más significativo de **I (R)** sea 1.

H Se pone a 0.

N Contendrá 0.

P/V Copia el estado de **IFF2**, es decir, 0 cuando las interrupciones están inhibidas y 1, cuando están habilitadas.

Muy bien, ¿pero qué quiere decir **IFF**? Entre los múltiples circuitos presentes en el **Z 80** dos son los encargados de señalarle cuando están habilitadas las interrupciones. Son denominados **IFF1**, el cual las permite o no, e **IFF2** que sirve de almacenamiento temporal del estado de **IFF1**. Las siglas **IFF** corresponden según su denominación inglesa a *Interruption enable Flip-Flop* y de ahí que en castellano normalmente se conoce como «bóscula de habilitación de interrupciones».

Cuando ejecutamos una instrucción **EI** (habilitar interrupción), **IFF1** e **IFF2** son puestas al mismo a 1. Al ser tratada ésta, simultáneamente pasan

a ser 0 y deberemos colocarlo por programa, nuevamente a 1, si queremos tratar otra interrupción. Todo esto será comentado con detalle en el capítulo dedicado a las interrupciones.

Otra característica a tener en cuenta se refiere a la periódica variación a que está sujeto el registro R (recordemos la exposición comentada anteriormente). Esta circunstancia puede aprovecharse en los programas en que necesitemos generar números aleatorios.

Si queremos comprobarlo introduzcamos en el cargador hexadecimal la siguiente línea y ejecutemos el programa ubicándolo a partir de la dirección 30000.

10 DATA "ED 5F 4F 06 00 C9 *",612**

Ahora sustruyamos la línea 10 por la siguiente:

10 PRINT USR 30000: PAUSE 0: GOTO 10

y volvemos a ejecutar RUN. Cada vez que pulsamos una tecla obtendremos en la pantalla el valor almacenado en R, al ejecutarse la rutina en C/M. Observaremos que éstos siempre están comprendidos entre 0 y 127, es decir, el bit de mayor peso del registro R siempre es 0.

Las instrucciones LD I,A y LD R,A introducen en los registros I o R, respectivamente, el contenido actual del acumulador añadiéndose de direccionamiento implícito. Con ésto completamos el recorrido efectuado a través del grupo de carga de 8 bits.



ENSAMBLADOR	CODIGO MAQUINA	Nº BYTES	CICLOS RELOJ	CICLOS MAQUINA
LD(X+d), r	1 1 0 1 1 1 0 1	3	19	5
	0 1 1 1 1 1 1 1			
	0 0 0 0 0 0 0 0			
LD(Y+d), r	1 1 1 1 1 1 0 1	3	19	5
	0 1 1 1 1 1 1 1			
	0 0 0 0 0 0 0 0			
LD(X+d), n	1 1 0 1 1 1 0 1	4	19	5
	0 0 1 1 0 1 1 0			
	0 0 0 0 0 0 0 0			
	0 0 0 0 0 0 0 0			
LD(Y+d), n	1 1 1 1 1 1 0 1	4	19	5
	0 0 1 1 0 1 1 0			
	0 0 0 0 0 0 0 0			
	0 0 0 0 0 0 0 0			
INDICADORES NO AFECTADOS				

MORSE



programa diferente

con MORSE podremos codificar todos los mensajes secretos que queramos y repetirlos hasta la saciedad. Es una buena forma de divertirse con un programa diferente.

EL PROGRAMA

Ante todo, es importante señalar que este programa utiliza unas llamadas a una subrutina en código máquina por lo que, antes de ejecutarlo, tras introducirlo en el Spectrum, es conveniente grabar el BASIC, con lo que evitaremos males mayores en descuidos inoportunos. Esta subrutina la hemos utilizado en anteriores programas para dar una mayor vistosidad a las pantallas. Se trata de la subrutina de caracteres gigantes de PSION COMPUTERS.

Relativo al uso de la citada rutina, hemos inclu-

Cuando el Spectrum haya finalizado la codificación irá apareciendo los sucesos de puntos y rayas que formen el mensaje.



Es importante saber que los mensajes deben ser siempre introducidos en minúsculas.

de en el programa que nos ocupe una ligera variación: la subrutina de caracteres gigantes trabaja con dos tipos diferentes de generadores de caracteres. El primero de ellos es el propio del ordenador almacenado en R.O.M. a partir de la dirección 15816. El segundo generador está almacenado en R.A.M. a continuación de 54016. Para poder acceder a este segundo generador, el Spectrum efectúa dos POKE a la Variable del Sistema CHARS (direcciones 23606 y 23607) para que el primero de dicha variable señale la nueva zona del generador de caracteres (256 bytes antes). Nada más ejecutar el programa, aparecerá en pantalla un menú con dos opciones: 1ª conversión alfabeto-morse y 2ª fin. Lógicamente pulsaremos la tecla 1 para ejercitarnos con el programa.

Es importante saber que los mensajes deben ser siempre introducidos en minúsculas. De entrada, el programa selecciona para el cursor el modo de minúsculas. No hay que alterarlo bajo ninguna circunstancia.

Una vez introducido el mensaje, debemos esperar unos pocos segundos para ver la nueva codificación a alfabeto Morse. El tiempo de espera está en función de la longitud del texto. Cuanto más largas, más codifica será nuestra espera. Cuando el Spectrum haya terminado la codificación, irá apareciendo los sucesos de puntos

y rayos, a la vez que entra en funcionamiento el altavoz del ordenador simultaneando con tonos cortos (puntos) y largos (rayos) la representación de paraiso.

ADOPCIÓN DEL PROGRAMA

Para la adopción del programa hemos de proceder en primer lugar a la grabación del soporte BASIC que se lista en el presente capítulo, grabando a continuación en la cinta dos bloques de memoria: el correspondiente al código máquina de la subrutina de caracteres gigantes y el nuevo generador de caracteres tipo DATA que apareció he-



Con MDR52, podremos codificar todos los mensajes secretos que queramos y repetirlas hasta la saciedad.

